11-1500

A

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of: Zheng Wang et al.
Title: METHODS FOR COMPARING VERSIONS OF A PROGRAM
Attorney Docket No.: 777.416US1

PATENT APPLICATION TRANSMITTAL

BOX PATENT APPLICATION
Commissioner for Patents
Washington, D.C. 20231

We are transmitting herewith the following attached items and information (as indicated with an "X"):

X Return postcard.
X Utility Patent Application under 37 CFR § 1.53(b) comprising:
X Specification (29 pgs, including claims numbered 1 through 46 and a 1 page Abstract).
X Formal Drawing(s) (6 sheets).
X Unsigned Combined Declaration and Power of Attorney (4 pgs).

The filing fee (NOT ENCLOSED) will be calculated as follows:

	No. Filed	No. Extra	Rate	Fee
TOTAL CLAIMS	46 - 20 =	26	x 18 =	**$468.00**
INDEPENDENT CLAIMS	9 - 3 =	6	x 80 =	**$480.00**
[ ] MULTIPLE DEPENDENT CLAIMS PRESENTED				**$0.00**
BASIC FEE				**$710.00**
TOTAL				**$1,658.00**

THE FILING FEE WILL BE PAID UPON RECEIPT OF THE NOTICE TO FILE MISSING PARTS.

SCHWEGMAN, LUNDBERG, WOESSNER & KLUTH, P.A.
P.O. Box 2938, Minneapolis, MN 55402 (612-373-6900)

By: _____
Atty: D. C. Peter Chu
Reg. No. 41,676

Customer Number 21186

"Express Mail" mailing label number: EL709305366US
Date of Deposit: November 14, 2000
This paper or fee is being deposited on the date indicated above with the United States Postal Service pursuant to 37 CFR 1.10, and is addressed to the Commissioner for Patents, Box Patent Application, Washington, D.C. 20231.

UNITED STATES PATENT APPLICATION

METHODS FOR COMPARING VERSIONS OF A PROGRAM

INVENTORS

Zeng Wang

Ken Pierce

Scott McFarling

Ramarath Venkatesan

Schwegman, Lundberg, Woessner, & Kluth, P.A.

1600 TCF Tower

121 South Eighth Street

Minneapolis, Minnesota 55402

ATTORNEY DOCKET 777.416US1

MICROSOFT 136607.1

DRAFTING ATTORNEY

Peter Chu

METHODS FOR COMPARING VERSIONS OF A PROGRAM

Technical Field

The technical field relates generally to program analysis. More particularly, it pertains to comparing one version of a program in binary format with another version of the program in binary format.

Copyright Notice - Permission

Background

A program is a list of statements. These statements are written by a programmer in a language that is readable by humans. This list of statements may be translated, through processes that include front-end compilation, to produce an executable file that can cause a computer to perform a desired action.

A program can be improved by using profile information. Profile information helps to characterize behaviors of a program. The effort to collect adequate profile information is rather time intensive. This situation is made worse when changes to a program require that profile information be collected all over again for the new version of the program. Not being able to collect profile information in a desired amount of time may lead to inferior programs and to the eventual lack of acceptance of such programs in the marketplace.

Thus, what is needed are systems, methods, and structures to reuse profile

information in multiple versions of a program.

Summary

Systems, methods, and structures for comparing versions of a program in binary format are discussed. An illustrative aspect includes a system for reusing profile information of a program. The system includes a comparator to define a match when a first value equals a second value. The system further includes a propagator to propagate profile information when a match is defined. And the system further includes a processing engine that processes a portion of a first version of the program to produce the first value and a portion of a second version of the program to produce the second value. The processing engine uses a set of information at a desired fuzziness level to produce the first value and the second value.

Another illustrative aspect includes a method for comparing versions of a program in binary format. The method includes finding equivalent contents in portions of two versions of the program. The method further includes finding equivalent structure in the portions of the two versions. And the method further includes forming a match when a portion of one of the two versions is an equivalence of a portion of the other of the two versions.

Another illustrative aspect includes a method for comparing versions of a program in binary format. The method includes finding equivalent procedures in a first version and a second version. The method further includes finding equivalent portions of data in equivalent procedures. And the method further includes finding equivalent portions of code in equivalent procedures.

Another illustrative aspect includes a method for comparing procedures in versions of a program. The method includes finding procedures having identical names, finding procedures by calculating one hash value based on the code, finding procedures having similar names, and finding procedures by comparing hash values for portions of procedures.

Another illustrative aspect includes a method for comparing data in a first version to a procedure in a second version of a program in binary format. The method includes finding equivalent portions of data using hash values and finding equivalent portions of data using positional information in the procedures.

Another illustrative aspect includes a method for comparing code in a procedure in a first version to a procedure in a second version of a program in binary format. The method includes finding equivalent portions of code using hash values and finding equivalent portions of code using control flow.

Another illustrative aspect includes a method for comparing code in versions of a program. The method includes hashing to form hash values from a set of information at a desired level. And the method further includes comparing the hash value of a portion of code in a first version to the hash value of a portion of code in a second version so as to define a match if the hash value of the portion of code in the first version equals the hash value of the portion of code in the second version.

Brief Description of the Drawings

Figure 1 is a system diagram that shows reusing profile information according to one aspect of the invention.

Figure 2 is a system diagram that shows an engine for comparing two versions of a program according to one aspect of the invention.

Figure 3 is a process diagram that shows a technique for comparing two versions of a program according to one aspect of the invention.

Figure 4 is a process diagram that shows a technique for comparing two versions of a program according to one aspect of the invention.

Figure 5 is a process diagram that shows a technique for comparing procedures between two versions of a program according to one aspect of the invention.

Figure 6 is a process diagram that shows a technique for comparing data between two versions of a program according to one aspect of the invention.

Figure 7 is a process diagram that shows a technique for comparing code between two versions of a program according to one aspect of the invention.

Figure 8 is a process diagram that shows a hashing technique for comparing two versions of a program according to one aspect of the invention.

Figure 9 is a table that shows various fuzziness levels that are used to create hash values according to one aspect of the invention.

Detailed Description

In the following detailed description of exemplary embodiments of the invention, reference is made to the accompanying drawings which form a part hereof, and in which is shown, by way of illustration, specific exemplary embodiments in which the invention may be practiced. In the drawings, like numerals describe substantially similar components throughout the several views. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention. Other embodiments may be utilized and structural, logical, electrical, and other changes may be made without departing from the spirit or scope of the present invention. The following detailed description is, therefore, not to be taken in a limiting sense, and the scope of the present invention is defined only by the appended claims.

Figure 1 is a system diagram according to one aspect of the invention. A system 100 includes a version 102 of a program. The version 102 can be considered a particular executable file in binary format from a compilation of the program. The system 100 includes a profile 106. The profile 106 includes information collected from the version 102.

The system 100 includes another version of the program, version 112. Version 112 includes updates to version 102. The updates include certain program changes 108.

After the version 112 is produced, new profile information may need to be collected. There are several problems associated with collecting new profile

information. First, the process of collecting profile information is time intensive; this hinders the ability to obtain desired profile information within a fixed amount of time. Second, the size and complexity of programs have increased with each generation of technology; this increases the need to obtain quality profile information to improve the programs. Third, the original development team, who has the expertise to collect profile information, may be dispersed after a program is shipped; this creates a void in the special knowledge needed to collect profile information when the program undergoes fixes over a period of time, such as several years.

Various embodiments of the invention solve these problems by reusing selected profile information, which was collected from other versions of the program, in the new version of the program. The various embodiments of the invention allow desired profile information, which is absent from other versions, to be collected for the new version. Thus, precious time can be focused on collecting the desired profile information for the new version of the program.

Returning to Figure 1, the system 100 includes a conversion 118. The conversion 118 compares the version 102 to the version 112. The conversion 118 selectively transfers profile information from profile 106 to profile 116. For that profile information, which is desired but is absent from the profile 116, the system 100 allows the collecting of additional profile information to be incorporated in the profile 116.

Figure 2 is a system diagram according to one aspect of the invention. A system 200 includes a first version 202_1 of a program and a second version 202_2 of the program. Both the first version 202_1 and the second version 202_2 are in binary format. Both the first version 202_1 and the second version 202_2 are input into the processing engine 204. The processing engine 204 processes a portion of the first version 202_1 to produce a value. The processing engine 204 also processes a portion of the second version 202_2 to produce another value. These values are used to determine whether profile information associated with the portion of the first

version 202_1 should be reused in the portion of the second version 202_2. The processing engine 204 iterates the processing for all remaining portions of the first versions 202_1 and all remaining portions of the second version 202_2.

The first version 202_1 and the second version 202_2 are input into an engine 204_1. The terms "first version" and "second version", as used hereinabove and hereinbelow, include versions that are created contemporaneously. In one embodiment, the term "first version" may be considered an older version and the term "second version" a newer version. The engine 204_1 compares a procedure in the first version 202_1 to find an equivalent procedure in the second version 202_2. The term "procedure" means the inclusion of functions, methods, or any chunks of code that are named. Each procedure in the first version 202_1 and the second version 202_2 is assigned a value. If a value for a procedure in the first version 202_1 equals a value for a procedure in the second version 202_2, then a match is formed. This match indicates that the procedure in the second version 202_2 is a candidate to reuse profile information from the profile information in the procedure in the first version 202_1.

If a match is found between two procedures, the matching process undergoes a refinement. The engine 204_2 compares a portion of data in the first version 202_1 to find an equivalent portion of data in the second version 202_1. Each portion of the data in the first version 202_1 and the second version 202_2 is assigned a value. The engine 204_2 uses this value and positional information of the portion of data to determine a match.

The engine 204_3 also helps to refine the match between two procedures. The engine 204_3 compares a portion of code in the procedure in the first version 202_1 to find an equivalent portion of code in the procedure in the second version 202_2. Each portion of the code in the procedures in the first version 202_1 and the second version 202_2 is assigned a value. The engine 204_3 uses this value and other information to determine a match. If there is a match, this also increases confidence that the procedure in the second version 202_2 is a candidate to reuse profile information.

The system 200 includes a comparator 206. The processing engine 204 uses the comparator 206 to define a match between two values. The system 200 includes a propagator 208. The propagator 208 propagates profile information from the first version 202_1 to the second version 202_2 when a match is defined. The system 200 also includes an annotator (not shown). The annotator annotates the strength of each match so as to enhance the propagation of profile information. In other words, a user can specify profile information to be propagated for those matches that are at a desired strength.

Figure 3 is a process diagram according to one aspect of the invention. A process 300 is a method for comparing multiple versions of a program in binary format. In reusing profile information, it is advantageous to determine whether portions of a first version of a program have equivalent portions in a second version of the program. Equivalent portions in the second version can be expected to execute in a manner similar to portions in the first version. Thus, profile information for the portions in the first version can be reused in the equivalent portions in the second version.

The process 300 helps to determine the equivalent portions. In various embodiments, the process 300 compares portions of a first version and a second version of a program in binary format. These portions include code blocks, which are basic blocks, and data blocks, which are defined according to how the data is accessed. Certain types of profile information are associated at a block level. Because the process 300 can compare portions of the first version and the second version in terms of blocks, propagation of profile information is made easier.

The process 300 includes an act 302 for finding equivalent contents in the first version and the second version of the program. Figures 5, 6, and 7 illustrate an implementation of the act 302 for finding equivalent contents. Such an implementation does not limit the embodiments of the invention, and other implementations may be used. Because most portions of the first version remain the same or undergo little change in the portions of the second version, the act 302

enhances the efficiency of the comparing process.

The process 300 includes an act 304 for finding equivalent structure in the first version and the second version of the program. The process 300 includes an act 306 for forming a match when a portion of one version is an equivalence of a portion of the other version. In one embodiment, the act of forming a match includes forming a match when the portion of one version executes in a manner similar to the portion of the other version. The process 300 also uses a fuzzy match for portions of the first and second versions that have small differences. For other blocks that do not match, the process 300 may selectively use the results from both the acts 302 and 304 to form a match.

Figure 4 is a process diagram according to one aspect of the invention. A process 400 is a method for comparing multiple versions of a program in binary format. When a software designer modifies the source code of a program, the software designer may cause several changes when the source code is compiled to a file in a binary format. Such changes include code being added, deleted, or moved within a procedure. Other changes include changes to the instructions, such as operands and opcode, names of procedures, and types. These changes are defined as direct changes. Other changes are defined as indirect changes. Indirect changes are caused by a shift in the address space. Such changes affect targets of control flow instructions, pointers, register allocation, and others. Indirect changes may hinder the process of comparing. The process 400 helps to recognize indirect changes so as to enhance the comparing process.

The process 400 includes an act 402 for finding equivalent procedures in a first version of a program and in a second version of the program. The act 402 performs the comparison by using information selected from a group consisting of name information, type information, and code information.

The process 400 controls the complexity of the analysis by focusing on analyzing procedures. The process 400 recognizes that changes from one version to the next are likely to occur within procedural boundaries, and thus it is advantageous

to start by looking at procedures in the various versions of the program.

The process 400 includes an act 404 for finding equivalent portions of data in equivalent procedures. The act 404 refines the act 402. . Figure 6 illustrates an implementation of the act 404 for finding equivalent portions of data. Such an implementation does not limit the embodiments of the invention, and other implementations may be used.

The process 400 includes an act 406 for finding equivalent portions of code in equivalent procedures. The act 406 finds equivalent portions of code between the first and the second versions by using a hash value. The hash value is calculated from the portions of code. The act 406 may also consider the ordering of the instructions in the calculation of the hash value. In one embodiment, the hash value is a 64-bit hash value, which is calculated for each basic block based on the opcodes and operands of the instructions within the basic block.

The act 406 finds equivalent portions of code by iterating the act of finding with different levels of matching fuzziness. This technique helps to deal with the indirect changes that can inject errors into the comparing process. Each level of matching fuzziness provides a desired tradeoff between being flexible to look past minor changes and being precise to identify correct matches. Figure 9 illustrates an implementation of the different levels of matching fuzziness. Such an implementation does not limit the embodiments of the invention, and other implementations may be used.

Figure 5 is a process diagram according to one aspect of the invention. A process 500 is a method for comparing procedures in multiple versions of a program. The process 500 includes an act 502 for finding procedures having identical names. The act 502 first looks for procedures with the identical extended name in a first version of the program and a second version of the program, and then the act 502 looks for procedures with the identical hierarchical name.

The term "extended name" means the name that is assigned by a compiler to a procedure that includes the hierarchical name, the parameter information, and

return type information. The term "hierarchical name" means the name that defines the entire class hierarchy. For example, if foo is a class and foo has a method bar, the hierarchical name of bar is foo::bar.

The process 500 includes an act 504 for finding procedures by calculating one hash value based on the code of the procedures. The act 504 is executed when procedures cannot be matched by name. The hash value is calculated for each procedure based on the code blocks of the procedure. The calculation of the hash value accounts for the order of code blocks within the procedure. The act 504 may be iterated using different levels of fuzziness. Figure 9 illustrates an implementation of the different levels of matching fuzziness. Such an implementation does not limit the embodiments of the invention, and other implementations may be used.

The process 500 includes an act 506 for finding procedures having similar names. The act 506 looks for procedures that have small differences in the hierarchical names. Then the act 506 calculates a hash value for each code block within the procedures. If the percentage is at a predetermined threshold for code blocks having equivalent hash values between the first and the second versions, the procedures are considered to be a match. A user may set the predetermined threshold.

The process 500 includes an act 508 for finding procedures by comparing hash values for portions of procedures between the first and the second versions. The act 508 calculates a hash value for each code block within the procedures. If the percentage is at a predetermined threshold for code blocks having equivalent hash values, the procedures can be considered to be a match. A user may set the predetermined threshold. This technique can define a match even when code blocks have been added or deleted from a procedure.

In one embodiment, the process 500 is executed in the following order: The act 502, the act 504, the act 506, and the act 508. However, the process 500 may be executed using any combination or sub-combination of acts 502, 504, 506, and 508.

Figure 6 is a process diagram according to one aspect of the invention. A

process 600 is a method for comparing data in a first version to a procedure in a second version of a program in binary format. The process 600 includes an act 602 for finding equivalent portions of data using hash values. The act 602 calculates a hash value for each portion of data. The act 602 then defines a match for each portion of data in the first version that has the same hash value as a portion of data in the second version.

The process 600 includes an act 604 to find equivalent portions of data using positional information in the procedures. The process 600 executes the act 604 for portions of data that cannot be matched using the act 602. In one embodiment, if portions of data are sandwiched by two pairs of matched portions of data, the process 600 defines them as a match. In one embodiment, the process 600 defines them as a match as long as their sizes do not differ by a predetermined threshold. The user may set the predetermined threshold.

Any technique for obtaining positional information may be used. In one embodiment, however, such positional information may be obtained as described in the following applications: U.S. SN 09/343,805 entitled "Translation and Transformation of Heterogeneous Programs", SN 09/343,298 entitled "Instrumentation and Optimization Tools for Heterogeneous Programs", SN 09/343,279 entitled "Shared Library Optimization for Heterogeneous Programs", SN 09/343,276 entitled "Application Program Interface for Transforming Heterogeneous Programs", and SN 09/343,287 entitled "Cross Module Representation of Heterogeneous Programs" (all filed on June 30, 1999). These applications do not limit the embodiments of the invention and will not be discussed here in full.

The result of process 600 can be used to enhance a subsequent comparing process. For example, if two instructions refer to two portions of data that are already matched to each other, the two instructions can be considered a match even if the data addresses are different.

Figure 7 is a process diagram according to one aspect of the invention. A

process 700 is a method for comparing code in a procedure in a first version to a procedure in a second version of a program in binary format.

The process 700 includes an act 702 for finding equivalent portions of code using hash values. In one embodiment, the act 702 finds equivalent basic blocks based on the code contents and the positions of the basic blocks. The act 702 identifies a one-to-one match between a basic block in the first version and a basic block in the second version. The act 702 can be iterated at a desired level of fuzziness to find a match. Figure 9 illustrates an implementation of the different levels of matching fuzziness. Such an implementation does not limit the embodiments of the invention, and other implementations may be used.

The process 700 includes an act 704 for finding equivalent portions of code using control flow. The process 700 selectively executes the act 704 for portions of code that cannot be matched by the act 702. The act 704 traverses down both versions of the procedure following the control flow. The act 704 uses conditional branches, jump instructions, return instructions, and previously matched blocks to pinpoint portions of code that can be considered equivalent in terms of control flow.

Figure 8 is a process diagram according to one aspect of the invention. A process 800 is a method for comparing code in multiple versions of a program in binary format. The process 800 includes an act 802 for hashing to form hash values from a set of information at a desired level. The desired level can be considered to be a fuzzy level. Each level defines the strength of a filter. If the filter is weak, the process 800 will generate more matches between portions of code in a first version and portions of code in a second version of a program. If the filter is strong, the process 800 will generate matches that are more likely to be accurate. The strength of the filter is based on a set of information that is included or excluded. This set of information will be discussed hereinbelow.

The process 800 includes an act 804 for comparing the hash value of a portion of code in the first version to the hash value of a portion of code in the second version. The act 804 defines a match if the hash value of the portion of code

in the first version equals the hash value of the portion of code in the second version. The process 800 includes iterating the act 802 to form hash values at another desired level and iterating the act 804 to compare the hash values. Figure 9 illustrates an implementation of the different levels of matching fuzziness. Such an implementation does not limit the embodiments of the invention, and other implementations may be used.

For certain fuzzy levels, there can be several portions of code with the same hash value. For procedures with the same hash values, the act 804 defines a match by the order of appearance of procedures in the program and also when a desired percentage of portions of code in the procedure in the first version matches the portions of code in a procedure in the second version.

For basic blocks with the same hash values, the act 804 executes a two-phase process when multiple basic blocks have the same hash value. The first phase of the two-phase process includes defining a match that selectively limits cross-matching. A cross-matching occurs when a new match crosses an existing match. The existing match is a match between a first basic block in the first version and a second basic block in the second version; imagine a first line that is drawn to connect the first and second basic blocks together. The new match is a match between a third basic block in the first version and a fourth basic block in the second version; imagine a second line that is drawn to connect the third and fourth basic blocks together. If the second line crosses the first line, then a cross-matching exists. As will be discussed hereinbelow, cross-matching is inhibited depending on the level of fuzziness.

The second phase of the two-phase process includes propagating a match between two portions of code to other portions of code in the vicinity of the two portions of code using a neighbor test. A pair of blocks passes the neighbor test if their predecessors or their successors are matched to each other. This new match can then be propagated to more successors or predecessors. This process can be repeated until the propagation stops.

The process 800 includes annotating each match to form information for

subsequent analysis. The information includes an indication of the strength of the match. This information can be used by a system to reuse profile information only for those matches that were matched at a desired strength.

Figure 9 is a table according to one aspect of the invention. Table 900 defines various fuzziness levels and the set of information that is included in the various fuzziness levels. Recall that the strength of the filter for matching between versions of a program is based on a set of information that is included or excluded.

The set of information includes numerical address offsets. Numerical address offsets are numerical offsets in memory address operands. These offsets often change from build to build due to changes in data layout.

The set of information includes register allocation. A minor code change may cause different results of register allocation for the rest of a procedure. This affects registers that are included in the allocation by the compiler.

The set of information includes immediate operands. The set of information also includes block address operands. Block address operands appear in control flow instructions, such as jump, branch, and call; and in others, such as pointer operations. An instruction that contains these operands is defined as a source instruction. A portion of code, such as a basic block, which is referred to by the operands, is defined as a target block. The term "target block" means the inclusion of target basic block or target instruction. Undesirable indirect changes affect these operands.

Thus, a special technique may be used to form hash values to account for these indirect changes. The technique includes calculating a hash value by treating the operand in a block in one version of the program the same as the operand in a block in the other version of the program when the target block has previously been matched to another block. The technique includes formulating a hash value based on the extended name of the target block. The technique includes figuring a hash value based on the offset of the address of the target block from the beginning of a procedure and the offset of the address of the target block from the source

instruction when the procedure contains both the target block and the source instruction. The technique includes determining a hash value based on the name of the procedure and the offset of the address of the target block from the beginning of the procedure when the procedure that contains the target block is different from another procedure that contains the source instruction. The technique may selectively execute the act of calculating, the act of formulating, the act of figuring, and the act of determining so as to inhibit errors arising in forming the hash value.

The set of information includes instruction opcode and operand types. For example, because of minor changes in the source code or the compilation process, a push word instruction may become a push double word instruction. As another example, a memory operand may become a register operand. As yet another example, a return instruction with no parameter may become a return with a parameter.

The set of information includes instructions that were added or removed. Again, the set of information forms the strength of the filter at various levels. Certain levels include more information to form the hash value. Other levels include less information to form the hash value. Certain levels allow accurate matches to be found for portions of code that have not changed or have only barely changed. Other levels allow matches to be found for portions of code that have changed considerably.

Returning to Figure 9, each fuzziness level is now discussed. Figure 9 illustrates an embodiment that has been implemented for processors that have X86 architecture. However, the invention is not limited to the X86 architecture and may include other processor architectures. Regarding level 0, all instruction opcodes and operands are included in the hashing. Level 0 can be used in procedure matching to find procedures that remain the same.

Regarding level 1, numerical address offsets are excluded from the hashing. Registers EAX, ECX, and EDX are converted to the same value for the calculation of the hash value. However, the dependency of these registers is retained in the

calculation of the hash value. Regarding level 1a, the set of information is the same as level 1. However, the hash value reflects only the last instruction in each portion of code.

Regarding level 2, the address offset of the target block from the beginning of a procedure is excluded. This accommodates indirect changes that cause address shift for part of a procedure.

Regarding level 3, all immediate operands and operands of return instructions are excluded from the hashing. Registers EAX, ECX, and EDX are converted to the same value for the hashing calculation. Separately, registers EBX, EDI, and ESI are converted to the same value for the hashing calculation. Register dependency information is not included. The matching status and extended name of target blocks are not included. The address offset from a source instruction to a target block in the same procedure is reduced to +1 or –1 based on the branch direction, such as whether forward or backward. Regarding level 3a, the set of information is the same as level 3. However, the hash value reflects only the last instruction in each portion of code.

Regarding level 4, each instruction is hashed based on the opcode and the types (but not the contents) of the operands. Regarding level 5, each instruction is hashed based on the opcode only. Certain groups of opcodes are considered to be the same, such as push word and push double word, all conditional branch opcodes, and others.

Recall the discussion on cross-matching hereinabove. In one embodiment, at fuzziness level 1a, cross-matching is forbidden. In another embodiment, at fuzziness level 3, cross-matching is forbidden for blocks with three instructions or less. In another embodiment, at fuzziness level 3a, the first phase of the two-phase process is skipped. In another embodiment, at fuzziness levels 4 and 5, no matching may occur for blocks with one or two instructions, and no cross-matching may occur for blocks with three instructions.

In the various embodiments, the order in which different fuzziness levels is

executed can be adjusted. Not all levels have to be used. Each level can be used more than once. In one embodiment, the comparing process performs the matching levels in increasing order of fuzziness. In another embodiment, the comparing process performs six passes during hashing-based procedure matching; this embodiment uses fuzziness levels in the order of 0, 1, 3, 1a, 5, and 3a. During basic block matching, the comparing process performs seven passes using fuzziness levels 1, 3, 2, 1a, 4, 3a, and 5.

Conclusion

Systems and methods have been described to reuse profile information in one version of a program in another version of a program. The techniques described hereinabove analyze the program in binary format without referring to changes made to the source code of the program. The techniques allow more time to collect profile information not found in the reused profile information.

Although the specific embodiments have been illustrated and described herein, it will be appreciated by those of ordinary skill in the art that any arrangement which is calculated to achieve the same purpose may be substituted for the specific embodiments shown. This application is intended to cover any adaptations or variations of the present invention. It is to be understood that the above description is intended to be illustrative, and not restrictive. Combinations of the above embodiments and other embodiments will be apparent to those of skill in the art upon reviewing the above description. The scope of the invention includes any other applications in which the above structures and fabrication methods are used. Accordingly, the scope of the invention should only be determined with reference to the appended claims, along with the full scope of equivalents to which such claims are entitled.

We claim:

1. A system for reusing profile information of a program, comprising:

a comparator to define a match when a first value equals a second value;

a propagator to propagate profile information when a match is defined; and

a processing engine that processes a portion of a first version of the program to produce the first value and a portion of a second version of the program to produce the second value, wherein the processing engine uses a set of information at a desired fuzziness level to produce the first value and the second value.

2. The system of claim 1, wherein the processing engine includes an engine to produce values for procedures.

3. The system of claim 1, wherein the processing engine includes an engine to produce values for portions of data.

4. The system of claim 1, wherein the processing engine includes an engine to produce values for portions of code within the procedures.

5. The system of claim 1, further comprising an annotator to annotate the strength of the match so as to enhance the propagation of profile information.

6. A method for comparing versions of a program in binary format, comprising:

finding equivalent contents in portions of two versions of the program;

finding equivalent structure in the portions of the two versions; and

forming a match when a portion of one of the two versions is an equivalence to a portion of the other of the two versions.

7. The method of claim 6, wherein the act of finding equivalent contents includes finding equivalent code in the portion of one version of the two versions and the portion of the other version of the two versions.

8. The method of claim 7, wherein the act of finding equivalent contents includes finding equivalent data in the portion of one version of the two versions and the portion of the other version of the two versions.

9. The method of claim 6, wherein the act of forming a match includes forming a match based on a fuzzy match for portions of the two versions that include small differences.

10. The method of claim 6, wherein the act of forming a match includes forming a match based on the results of both the act of finding equivalent contents and the

act of finding equivalent structure.

11. A computer readable medium having instructions stored thereon for causing a computer to perform a method for comparing versions of a program in binary format, the method comprising:

finding equivalent contents in portions of two versions of the program;

finding equivalent structure in the portions of the two versions; and

forming a match when a portion of one of the two versions is an equivalence to a portion of the other of the two versions.

12. The method of claim 11, wherein the act of finding equivalent contents includes finding equivalent code in the portion of one version of the two versions and the portion of the other version of the two versions.

13. The method of claim 12, wherein the act of finding equivalent contents includes finding equivalent data in the portion of one version of the two versions and the portion of the other version of the two versions.

14. The method of claim 11, wherein the act of forming a match includes forming a match based on a fuzzy match for portions of the two versions that include small differences.

15. The method of claim 11, wherein the act of forming a match includes forming a match based on the results of both the act of finding equivalent contents and the act of finding equivalent structure.

16. A method for comparing versions of a program in binary format, comprising:

finding equivalent procedures in a first version and a second version;

finding equivalent portions of data in equivalent procedures; and

finding equivalent portions of code in equivalent procedures.

17. The method of claim 16, wherein the act of finding equivalent procedures includes comparing information selected from a group consisting of name information, type information, and code information.

18. The method of claim 16, wherein the act of finding equivalent portions of code includes finding equivalent portions of code based on a hash value, which is calculated from the instructions in the portions of code and the ordering of the instructions.

19. The method of claim 18, wherein the act of finding equivalent portions of code based on a hash value includes iterating the act of finding with different levels of matching fuzziness so as to enhance the act of finding equivalent portions of code.

20. The method of claim 16, wherein the method executes a desired combination of at least one of the act of finding the equivalent procedure, the act of finding equivalent portions of data, and the act of finding equivalent portions of code.

21. A method for comparing procedures in versions of a program, comprising:

finding procedures having identical names;

finding procedures by calculating one hash value based on the code;

finding procedures having similar names; and

finding procedures by comparing hash values for portions of procedures.

22. The method of claim 21, wherein the method executes a desired combination of acts, wherein the desired combination includes at least one of the act of finding procedures having identical names, the act of finding procedures by calculating a hash value based on the code, the act of finding procedures having similar names, and the act of finding procedures by comparing hash values for portions of procedures.

23. The method of claim 21, wherein the act of finding procedures having identical names includes executing at least one of an act of finding procedures having identical extended names and an act of finding procedures having identical hierarchical names.

24. The method of claim 21, wherein the act of finding procedures by calculating a hash value based on the code includes finding procedures that lack identical names by calculating the hash value that considers the order of the code in the procedures.

25. The method of claim 21, wherein the act of finding procedures having similar names includes finding procedures having small differences in the hierarchical names and having a high percentage of blocks between the procedures that have equivalent hash values.

26. The method of claim 21, wherein the act of finding procedures by comparing hash values for portions of procedures includes finding procedures, which cannot be matched by names, by comparing whether a hash value for blocks within one procedure is equivalent to a hash value for blocks within another procedure.

27. A method for comparing data in a first version to a second version of a program in binary format, comprising:

finding equivalent portions of data using hash values; and

finding equivalent portions of data using positional information in the procedures.

28. The method of claim 27, wherein the method executes a desired combination of acts, wherein the desired combination includes at least one of the act of finding equivalent portions of data using hash values and the act of finding equivalent portions of data using positional information in the procedures.

29. The method of claim 27, wherein finding equivalent portions of data using positional information includes defining that a portion of data in the first version matches a portion of data in the second version when the portion of data sandwiches between previously matched portions of data.

30. A method for comparing code in a procedure in a first version to a procedure in a second version of a program in binary format, comprising:

finding equivalent portions of code using hash values; and

finding equivalent portions of code using control flow.

31. The method of claim 30, wherein the method executes a desired combination of acts, wherein the desired combination includes at least one of the act of finding equivalent portions of code using hash values and the act of finding equivalent portions of code using control flow.

32. The method of claim 30, wherein the act of finding equivalent portions of code using hash values includes calculating hash values based on code contents and

positional information.

33. The method of claim 30, wherein the act of finding equivalent portions of code using hash values is iterated, wherein each iteration uses a desired level of fuzziness to define a match between a portion of code in the procedure in the first version and a portion of code in the procedure in the second version of code.

34. The method of claim 30, wherein the act of finding equivalent portions of code using control flow selected from a group consisting of conditional branches, jump instructions, return instructions, and previously matched portions of code so as to define that a portion of code in the procedure in the first version matches a portion of code in the procedure in the second version.

35. A method for hashing code to compare versions of a program, comprising:

defining an instruction that contains an operand, which appears in a control flow, as a source instruction, and defining the block referred to by the operand as a target block; and

forming a hash value, wherein forming includes calculating the hash value by treating the operand in a block in one version of the program the same as the operand in a block in the other version of the program when the target block has previously been matched to another block.

36. The method of claim 35, wherein forming includes formulating the hash value based on the extended name if the target block already has a compiler-assigned extended name.

37. The method of claim 36, wherein forming includes figuring the hash value based on the offset of the address of the target block from the beginning of a procedure and the offset of the address of the target block from the source instruction when the procedure contains the target block and the source instruction.

38. The method of claim 37, wherein forming includes determining the hash value based on the name of a procedure and the offset of the address of the target block from the beginning of the procedure when the procedure that contains the target block is different from another procedure that contains the source instruction.

39. The method of claim 38, wherein the method selectively executes the act of calculating, the act of formulating, the act of figuring, and the act of determining so as to inhibit errors from arising in forming the hash value.

40. A method for comparing code in versions of a program, comprising:

hashing to form hash values from a set of information at a desired level; and

comparing the hash value of a portion of code in the first version to the hash value of a portion of code in the second version so as to define a match if the hash

value of the portion of code in the first version equals the hash value of the portion of code in the second version.

41. The method of claim 40, further comprising iterating the method, wherein iterating includes iterating the act of hashing to form hash values from another set of information of another desired level.

42. The method of claim 40, wherein comparing includes comparing the hash value of a procedure in the first version to the hash value of a procedure in the second version, wherein the act of comparing defines a match by the order of appearance of procedures in the program when multiple procedures have the same hash value.

43. The method of claim 42, wherein comparing includes comparing the hash value of each portion of code in a procedure in the first version to the hash value of each portion of code in a procedure in the second version, wherein the act of comparing defines a match when a desired percentage of portions of code in the procedure in the first version matches the portions of code in the second version.

44. The method of claim 40, wherein comparing includes comparing the hash value of a portion of code in a procedure in the first version to the hash value of a portion of code in a procedure in the second version, wherein the act of comparing

executes a two-phase process when multiple portions of code have the same hash value.

45. The method of claim 40, wherein comparing includes executing a two-phase process, wherein the first phase of the two-phase process includes forming a match that selectively limits cross-matching, and wherein the second phase of the two-phase process includes propagating a match between two portions of code to other portions of code in the vicinity of the two portions of code using a neighbor test.

46. The method of claim 40, further comprising annotating each match to form information for subsequent analysis.

METHODS FOR COMPARING VERSIONS OF A PROGRAM

Abstract of the Disclosure

Systems and methods are discussed that allow profile information to be reused by various versions of a program. One illustrative aspect includes a method for comparing versions of a program in binary format. The method includes finding equivalent contents in portions of two versions of the program, finding equivalent structure in the portions of the two versions, and forming a match when a portion of one of the two versions is an equivalence of a portion of the other of the two versions.

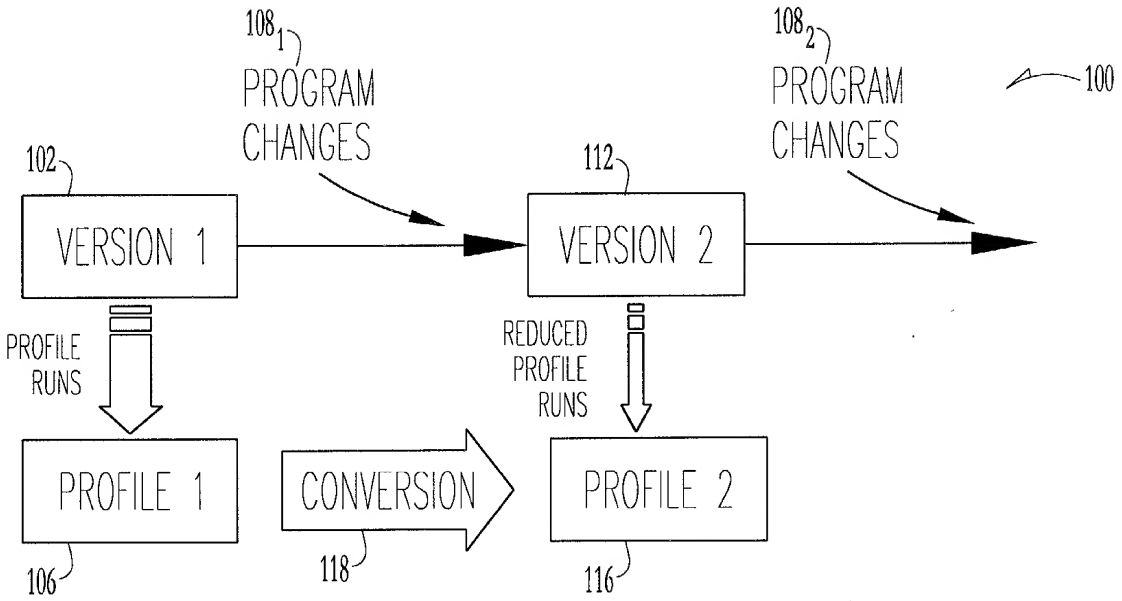

Fig.1

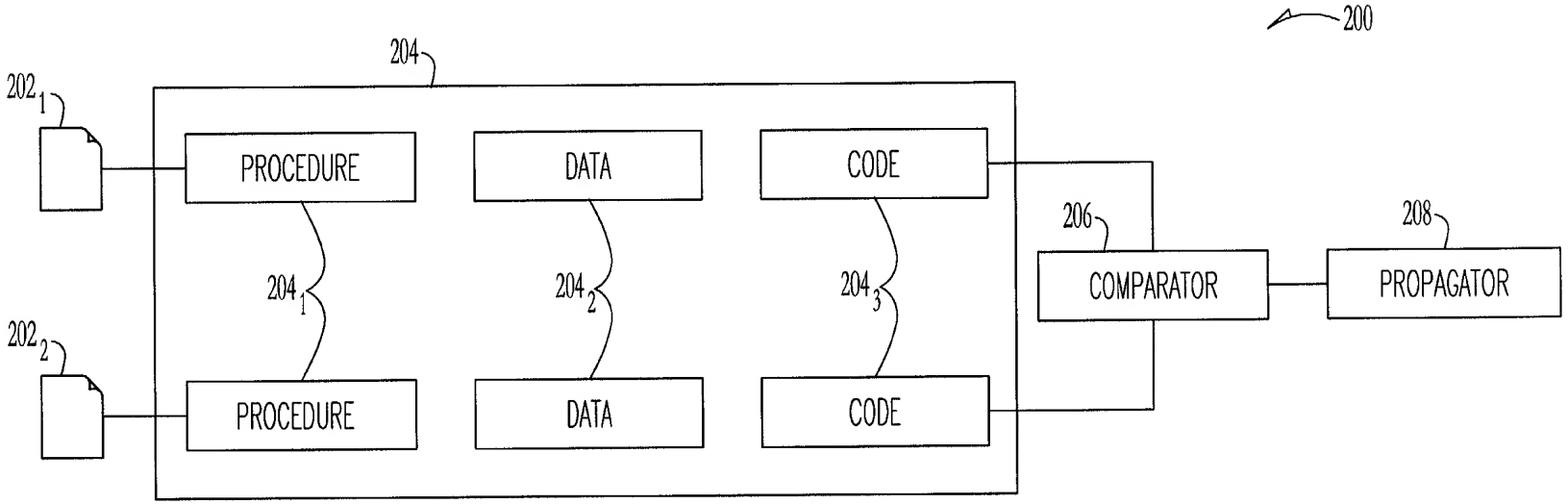

Fig.2

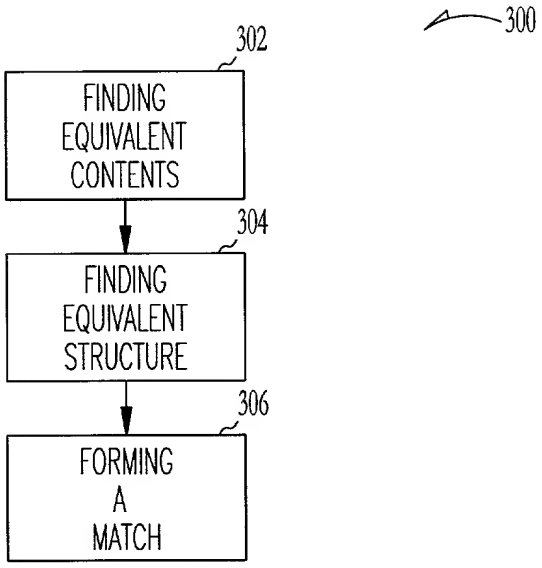
300
302
FINDING EQUIVALENT CONTENTS
304
FINDING EQUIVALENT STRUCTURE
306
FORMING A MATCH

Fig.3

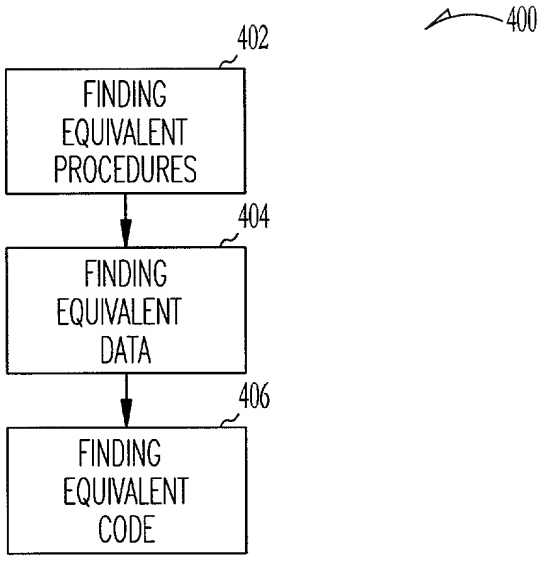
400
402
FINDING EQUIVALENT PROCEDURES
404
FINDING EQUIVALENT DATA
406
FINDING EQUIVALENT CODE

Fig.4

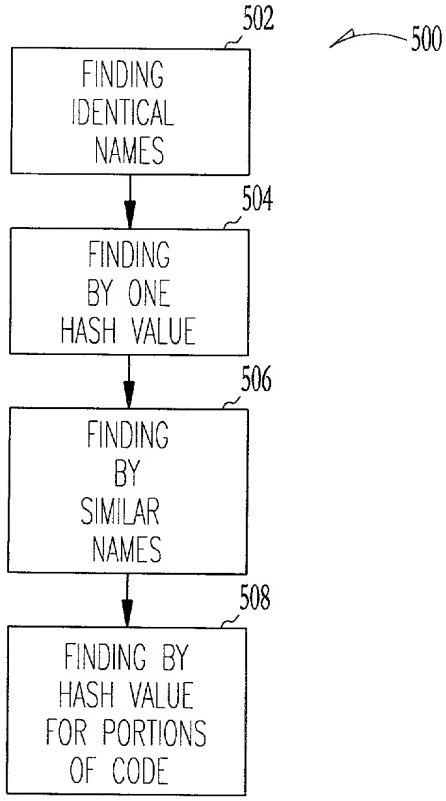
500
502
FINDING IDENTICAL NAMES
504
FINDING BY ONE HASH VALUE
506
FINDING BY SIMILAR NAMES
508
FINDING BY HASH VALUE FOR PORTIONS OF CODE

Fig.5

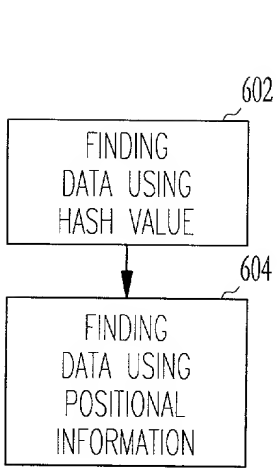
600
602
FINDING DATA USING HASH VALUE
604
FINDING DATA USING POSITIONAL INFORMATION

Fig.6

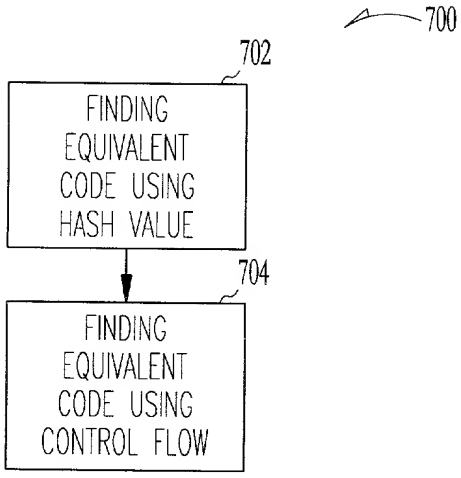
700
702
FINDING EQUIVALENT CODE USING HASH VALUE
704
FINDING EQUIVALENT CODE USING CONTROL FLOW

Fig.7

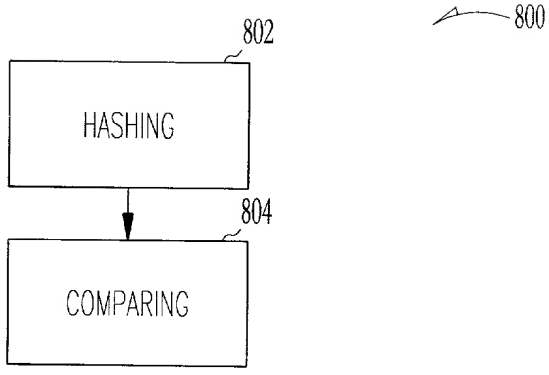
800
802
HASHING
804
COMPARING

Fig.8

900

FUZZINESS LEVEL	NUMERICAL ADDRESS OFFSET	REGISTER ALLOCATION	BLOCK ADDRESS OPERAND	OPERAND	OPCODE
0	ALL	ALL	•TARGET BLOCK'S MATCH •TARGET BLOCK'S EXTENDED NAME •TARGET PROCEDURE NAME OR BRANCH OFFSET WITHIN PROCEDURE •TARGET BLOCK'S DISTANCE FROM BEGINNING OF PROCEDURE	ALL	ALL
1	NONE	EAX/ECX/EDX: DEPENDENCY ONLY	SAME AS LEVEL 0	ALL	ALL
1A	SAME AS LEVEL 1, BUT INCLUDES ONLY THE LAST INSTRUCTION IN EACH BLOCK				
2	NONE	EAX/ECX/EDX: DEPENDENCY ONLY	•TARGET BLOCK'S MATCH •TARGET BLOCK'S EXTENDED NAME •TARGET PROCEDURE NAME OR BRANCH OFFSET WITHIN PROCEDURE	ALL	ALL
3	NONE	EAX=ECX=EDX EBX=EDI=ESI	TARGET PROCEDURE NAME OR BRANCH DIRECTION WITH PROCEDURE	NO IMMEDIATE NONE FOR RETURN	ALL
3A	SAME AS LEVEL 3, BUT INCLUDES ONLY THE LAST INSTRUCTION IN EACH BLOCK				
4	NONE	N/A	N/A	TYPE ONLY	ALL
5	NONE	N/A	N/A	NONE	GROUP

Fig.9

SCHWEGMAN ■ LUNDBERG ■ WOESSNER ■ KLUTH

United States Patent Application

COMBINED DECLARATION AND POWER OF ATTORNEY

As a below named inventor I hereby declare that: my residence, post office address and citizenship are as stated below next to my name; that

I verily believe I am the original, first and joint inventor of the subject matter which is claimed and for which a patent is sought on the invention entitled: **METHODS FOR COMPARING VERSIONS OF A PROGRAM**.

The specification of which is attached hereto.

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the patentability of this application in accordance with 37 C.F.R. § 1.56 (attached hereto). I also acknowledge my duty to disclose all information known to be material to patentability which became available between a filing date of a prior application and the national or PCT international filing date in the event this is a Continuation-In-Part application in accordance with 37 C.F.R. §1.63(e).

I hereby claim foreign priority benefits under 35 U.S.C. §119(a)-(d) or 365(b) of any foreign application(s) for patent or inventor's certificate, or 365(a) of any PCT international application which designated at least one country other than the United States of America, listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on the basis of which priority is claimed:

No such claim for priority is being made at this time.

I hereby claim the benefit under 35 U.S.C. § 119(e) of any United States provisional application(s) listed below:

No such claim for priority is being made at this time.

I hereby claim the benefit under 35 U.S.C. § 120 or 365(c) of any United States and PCT international application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT international application in the manner provided by the first paragraph of 35 U.S.C. § 112, I acknowledge the duty to disclose material information as defined in 37 C.F.R. § 1.56(a) which became available between the filing date of the prior application and the national or PCT international filing date of this application:

No such claim for priority is being made at this time.

I hereby appoint the following attorney(s) and/or patent agent(s) to prosecute this application and to transact all business in the Patent and Trademark Office connected herewith:

Anglin, J. Michael	Reg. No. 24,916	Hill, Stanley K.	Reg. No 37,548	Nelson, Albin J	Reg No 28,650
Beekman, Marvin L.	Reg. No. 38,377	Huebsch, Joseph C.	Reg. No. 42,673	Nielsen, Walter W.	Reg. No. 25,539
Bianchi, Timothy E.	Reg. No. 39,610	Jurkovich, Patti J.	Reg. No. 44,813	Oh, Allen J.	Reg. No. 42,047
Billion, Richard E.	Reg. No. 32,836	Kalis, Janal M.	Reg. No. 37,650	Padys, Danny J.	Reg No. 35,635
Black, David W.	Reg. No. 42,331	Kaufmann, John D.	Reg. No. 24,017	Parker, J. Kevin	Reg. No. 33,024
Brennan, Leoniede M.	Reg. No. 35,832	Klima-Silberg, Catherine I	Reg. No. 40,052	Perdok, Monique M.	Reg. No. 42,989
Brennan, Thomas F.	Reg. No. 35,075	Kluth, Daniel J	Reg. No. 32,146	Prout, William F	Reg No. 33,995
Brooks, Edward J , III	Reg. No 40,925	Lacy, Rodney L.	Reg. No. 41,136	Sako, Katie E.	Reg. No. 32,628
Chu, Dinh C.P.	Reg. No. 41,676	Lemaire, Charles A.	Reg. No. 36,198	Schumm, Sherry W.	Reg. No. 39,422
Clark, Barbara J.	Reg. No. 38,107	LeMoine, Dana B.	Reg. No. 40,062	Schwegman, Micheal L.	Reg. No. 25,816
Clise, Timothy B.	Reg. No. 40,957	Lundberg, Steven W	Reg. No. 30,568	Scott, John C.	Reg. No. 38,613
Crouse, Daniel D	Reg. No. 32,022	Maeyaert, Paul L.	Reg. No. 40,076	Smith, Michael G.	Reg. No. 45,368
Dahl, John M.	Reg. No. 44,639	Maki, Peter C.	Reg. No 42,832	Speier, Gary J.	Reg. No. 45,458
Drake, Eduardo E.	Reg. No. 40,594	Malen, Peter L.	Reg. No. 44,894	Steffey, Charles E.	Reg. No. 25,179
Embretson, Janet E	Reg. No. 39,665	Mates, Robert E.	Reg. No. 35,271	Terry, Kathleen R.	Reg. No. 31,884
Fordenbacher, Paul J.	Reg. No. 42,546	McCrackin, Ann M.	Reg No. 42,858	Tong, Viet V.	Reg. No. 45,416
Forrest, Bradley A	Reg. No. 30,837	Moore, Charles L , Jr.	Reg. No. 33,742	Viksnins, Ann S.	Reg. No. 37,748
Gamon, Owen J.	Reg. No. 36,143	Nama, Kash	Reg. No. 44,255	Woessner, Warren D.	Reg. No. 30,440
Harris, Robert J.	Reg. No. 37,346				

I hereby authorize them to act and rely on instructions from and communicate directly with the person/assignee/attorney/firm/organization/who/which first sends/sent this case to them and by whom/which I hereby declare that I have consented after full disclosure to be represented unless/until I instruct Schwegman, Lundberg, Woessner & Kluth, P.A. to the contrary.

Please direct all correspondence in this case to **Schwegman, Lundberg, Woessner & Kluth, P.A.** at the address indicated below:

P.O. Box 2938, Minneapolis, MN 55402
Telephone No. (612)373-6900

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of joint inventor number 1 : **Zheng Wang**
Citizenship: **Peoples Republic of China** Residence: **Cambridge, MA**
Post Office Address: 33 Oxford St., MD309
Cambridge, MA 02138

Signature: _____ Date: _____
Zheng Wang

Full Name of joint inventor number 2 : **Scott A. McFarling**
Citizenship: **United States of America** Residence: **Redmond, WA**
Post Office Address: 15934 NE 83rd Way
Redmond, WA 98052

Signature: _____ Date: _____
Scott A. McFarling

X Additional inventors are being named on separately numbered sheets, attached hereto.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of joint inventor number 3 : **Ken B. Pierce**
Citizenship: **United States of America** Residence: **Seattle, WA**
Post Office Address: 7920 Wallingford Avenue
#202
Seattle, WA 98103

Signature: _____ Date: _____

Ken B. Pierce

Full Name of joint inventor number 4 : **Ramarathnam Venkatesan**
Citizenship: **India** Residence: **Redmond, WA**
Post Office Address: 17208 NE 22nd Ct
Redmond, WA 98052

Signature: _____ Date: _____

Ramarathnam Venkatesan

Full Name of inventor:
Citizenship: Residence:
Post Office Address:

Signature: _____ Date: _____

Full Name of inventor:
Citizenship: Residence:
Post Office Address:

Signature: _____ Date: _____

§ 1.56 Duty to disclose information material to patentability.

(a) A patent by its very nature is affected with a public interest. The public interest is best served, and the most effective patent examination occurs when, at the time an application is being examined, the Office is aware of and evaluates the teachings of all information material to patentability. Each individual associated with the filing and prosecution of a patent application has a duty of candor and good faith in dealing with the Office, which includes a duty to disclose to the Office all information known to that individual to be material to patentability as defined in this section. The duty to disclose information exists with respect to each pending claim until the claim is canceled or withdrawn from consideration, or the application becomes abandoned. Information material to the patentability of a claim that is canceled or withdrawn from consideration need not be submitted if the information is not material to the patentability of any claim remaining under consideration in the application. There is no duty to submit information which is not material to the patentability of any existing claim. The duty to disclose all information known to be material to patentability is deemed to be satisfied if all information known to be material to patentability of any claim issued in a patent was cited by the Office or submitted to the Office in the manner prescribed by §§ 1.97(b)-(d) and 1.98. However, no patent will be granted on an application in connection with which fraud on the Office was practiced or attempted or the duty of disclosure was violated through bad faith or intentional misconduct. The Office encourages applicants to carefully examine:

(1) prior art cited in search reports of a foreign patent office in a counterpart application, and

(2) the closest information over which individuals associated with the filing or prosecution of a patent application believe any pending claim patentably defines, to make sure that any material information contained therein is disclosed to the Office.

(b) Under this section, information is material to patentability when it is not cumulative to information already of record or being made of record in the application, and

(1) It establishes, by itself or in combination with other information, a prima facie case of unpatentability of a claim; or

(2) It refutes, or is inconsistent with, a position the applicant takes in:

(i) Opposing an argument of unpatentability relied on by the Office, or

(ii) Asserting an argument of patentability.

A prima facie case of unpatentability is established when the information compels a conclusion that a claim is unpatentable under the preponderance of evidence, burden-of-proof standard, giving each term in the claim its broadest reasonable construction consistent with the specification, and before any consideration is given to evidence which may be submitted in an attempt to establish a contrary conclusion of patentability.

(c) Individuals associated with the filing or prosecution of a patent application within the meaning of this section are:

(1) Each inventor named in the application:

(2) Each attorney or agent who prepares or prosecutes the application; and

(3) Every other person who is substantively involved in the preparation or prosecution of the application and who is associated with the inventor, with the assignee or with anyone to whom there is an obligation to assign the application.

(d) Individuals other than the attorney, agent or inventor may comply with this section by disclosing information to the attorney, agent, or inventor.